Assignment – 4

Project Report

By,
Raghuveer Krishnamurthy Kanchibail *(rkanchib@iu.edu)*
Supreeth Keragodu Suryaprakash *(skeragod@indiana.edu)*
Suhas Jagadish *(jagadiss@iu.edu)*

Part 1: Creating stereograms in class we will distribute glasses having a red lens (to be worn on your left eye, which is your right eye from the perspective of someone looking at you) and a cyan lens (for your right eye). These are a simple form of 3d viewing technology, and the mechanism is straightforward: since cyan and red are complementary colors, pixels in the red channel will not be (very) visible to the right eye, and vice-versa. To create the illusion of 3d, we can create special images that encode two separate images, each of which is visible to one eye. Figure 1 shows a simple way of encoding a 3d image. The left image of the figure is what the image looked like from one of the two cameras that captured the stereogram. The other is a disparity map, telling you how far apart that pixel was across the two cameras. Recall that disparity and depth are inversely related. 1 Figure 1: An image and corresponding disparity map, where brightness is inversely proportional to depth. Write a program that accepts an image and a disparity map, like this: ./render image.png disp.png and generates an image that appears to be in 3d when viewed through the colored glasses. We've included a few test image pairs to get you started.

Solution:

We performed this task by making use of a temporary image. We'll calculate disparity value by the below equation ceil(pixel_of_disparity_image * 0.05), where we take each pixel of the disparity image and multiply it by a constant factor of 0.05 to calculate the disparity and width wise we calculate the shift by the original image.

Only if the shift is greater than 0, we'll allocate the pixels to a new image. In the second multiple for loops, we'll take only the first channel of the temporary image and assign it to the resultant image with remaining channels being the same.

By this we are generating the 3D images. The program for this takes a total time of O(image_width * image_height) to execute.

How to Run the Code?

Please Use this Command in order to run the program.

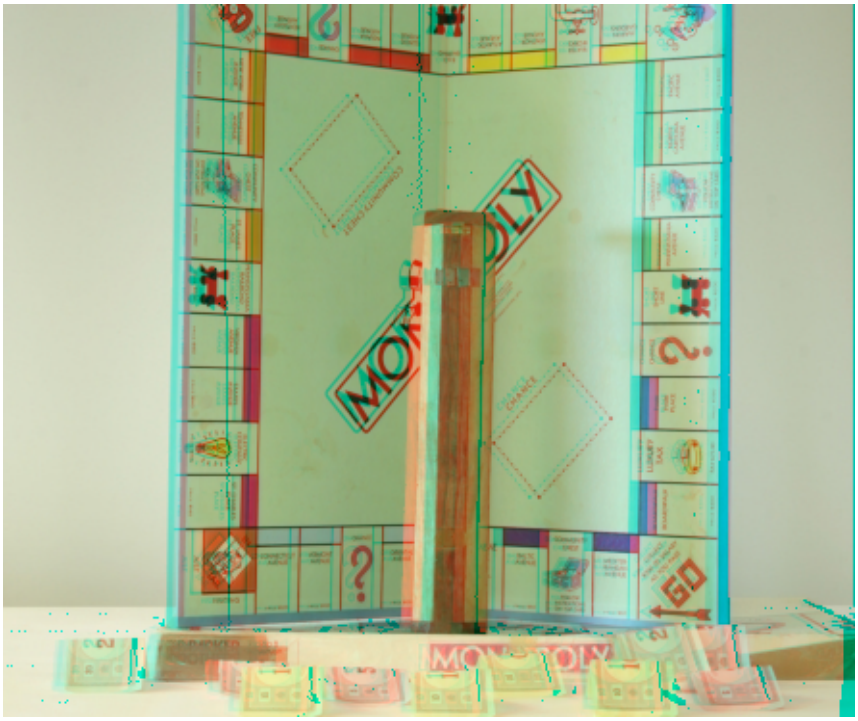
```
./render <image1.ext> <image2.ext>
```

e.g. `./render part1/birds/view1.png part1/birds/disp1.png`

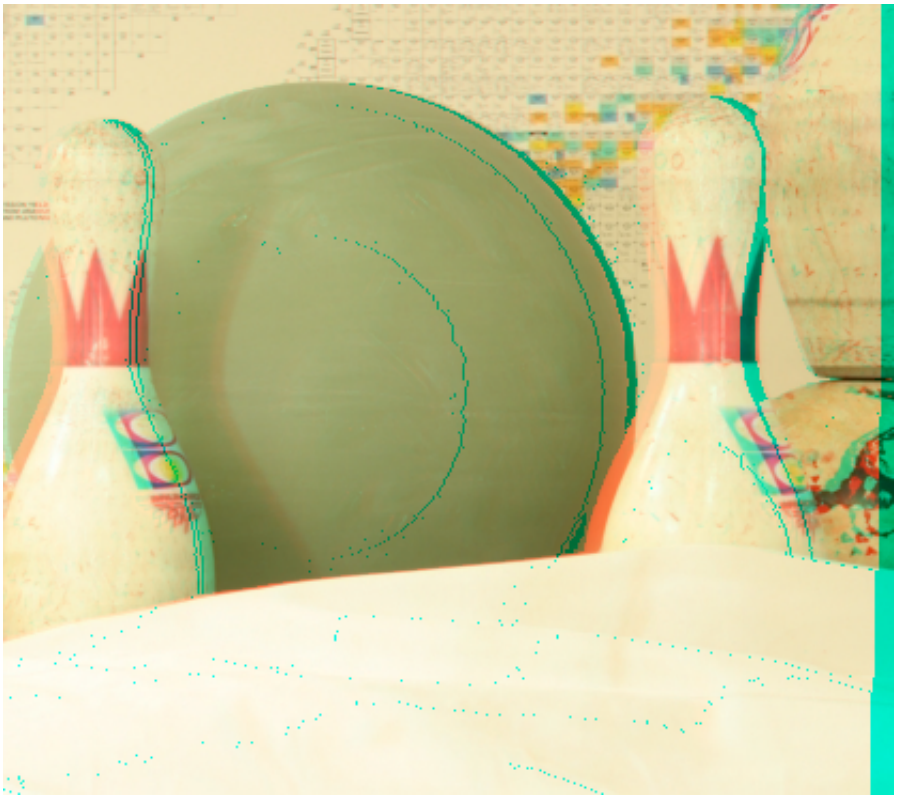
The results look like this,

Birds:

Monopoly:



Bowling:



Part 2: Background-foreground segmentation What if we'd like to display an image in 3d, but only have a single image (not a pair) and no depth map? This is an active area of research. As a starting point, let's say you want to create a stereo image that has just two depths: a foreground object or region, and a background region. To do this, of course, we'll first have to segment the image into the two regions. Let's use a semi-supervised approach. Our program will take a color image and a set of "seed points" that indicate where some of the foreground and background pixels in the image are located.

Solution:

We created an image called 'result' with the dimension same as that of the input image. Then, for all the points in the foreground we assigned '1' and for all the points present in the background we assigned '0'. We then proceeded to calculate the mean of all the points present in the foreground for all the 3 channels.

Similarly, we also found the covariance. Then we calculated the determinant of the inverse of covariance. All these values were used in the calculation of the gaussian probability density function. We iterated through each pixel of the image and assigned a value of '1' for a pixel present in foreground and '0' for a pixel present in background in the result image. For pixel

present in neither of them, we calculated the cost using the formula -log(Gaussian probabilty). The resulting cost was compared with β which was set to a value of '-4'.

If the cost was less than 'beta' the corresponding pixel of the input image in the result image was marked as '1' otherwise '0'.

How to Run the Code?

```
./segment <image1> <image2>
```

e.g. `./segment image1.png image2.png`

where <image1> – Input Image and <image2> – Seed Image.

Fig 1. Detecting Background

Fig 2. Detecting Foreground

Part 2.2

We implemented the loopy belief propagation. We applied BP rules in spite of loops. In each iteration, each node sends all messages in parallel.

The drawback is that it may not converge. There are couple of techniques which are present to handle it, but we couldn't perform because of the lack of time.

We ran our code for 10 iterations with value of α being 10 and λ being 100.

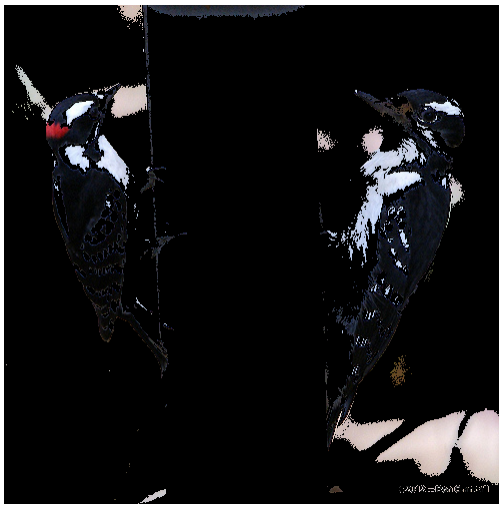
We tried testing it for different thresholds, Thresholds did change for each of the images. The α and λ varied for each images which made it difficult to find the proper one. This was the result we got.

How to run the Code?

```
./segment <image1> <image2>
```

e.g. `./segment image1.png image2.png`

Background



Foreground

Part 2.3

In this task, we need to create 3d images by using foreground and background images. We used the similar function we wrote to perform the operation on the first question.

The results were rather good. You can view it here.

How to Run the Code?

`./segment <image1> <image2>`

e.g. `./segment image1.png image2.png`

Say this generate image_fg.png and image_bg.png

`./render <image1.ext> <image2_background.ext>`

e.g. `./render image1.png image_bg.png`

Part 3: